FAQ

This **FAQ** will answer some basic questions for developers building a game on the endless template game.

Missions

How can I add new Missions?

Missions are created in the class `MissionProvider.m`. The template game creates for sample missions, using three mission types:

```
RunDistanceMission *mission1 = [[RunDistanceMission alloc] init];
    mission1.missionDescriptionFormat = @"Run %d meters";
    mission1.thumbnailFileName = @"missions_1.png";
    [[mission1.missionObjectives objectAtIndex:0] setGoalValue:50];
    [self storeLastObjectiveGoalValuesForMission:mission1];

    KillAmountOfEnemiesMission *mission2 = [[KillAmountOfEnemiesMission
alloc] init];
    mission2.missionDescriptionFormat = @"Kill %d enemies";
    mission2.thumbnailFileName = @"missions_2.png";
    [[mission2.missionObjectives objectAtIndex:0] setGoalValue:1];
    [self storeLastObjectiveGoalValuesForMission:mission2];

    KillAmountOfEnemiesInDistance *mission3 =
[[KillAmountOfEnemiesInDistance alloc] init];
    mission3.missionDescriptionFormat = @"Kill %d enemies in %d meters";
    mission3.thumbnailFileName = @"missions_1.png";
    [[mission3.missionObjectives objectAtIndex:0] setGoalValue:2];
    [[mission3.missionObjectives objectAtIndex:1] setGoalValue:1500];
    [self storeLastObjectiveGoalValuesForMission:mission3];

    RunDistanceMission *mission4 = [[RunDistanceMission alloc] init];
    mission4.missionDescriptionFormat = @"Run %d meters";
    mission4.thumbnailFileName = @"missions_1.png";
    [[mission4.missionObjectives objectAtIndex:0] setGoalValue:150];
    [self storeLastObjectiveGoalValuesForMission:mission4];

    [missions addObjectsFromArray:@[mission1, mission2, mission3,
mission4]];
```

Missions consist of 1 or 2 objectives. At the moment the template game has three mission types:

- RunDistanceMission
    - 1 Objective: RunDistance (HigherIsBetter)

- KillAmountOfEnemiesMission
    - 1 Objective: KilledAmountEnemies (HigherIsBetter)
- KillAmountOfEnemiesInDistance
    - 2 Objectives:
        - KilledAmountEnemies (HigherIsBetter)
        - Distance (LowerIsBetter)

If you simply want to add different variations of these existing Mission-Types, you can create new instances and change the goal values for the objectives.

All missions added to the `missions` array, will be played by the player in that order. The player will always see 3 missions, once one is completed, the next one from the missions array is picked.

When the end of the missions array is reached, the `MissionProvider` will generate missions, based on the previous ones in the `missions` array.

Whenever you create a mission manually, it is useful to call `[self storeLastObjectiveGoalValuesForMission:missionX];`.

That method will store the highest difficulty for each mission type. That information is used to generate new missions, that are more difficult than previous ones. That is why, objectives can have to different `objectiveTypes`:

```
typedef NS_ENUM(NSInteger, MissionObjectiveType) {
MissionObjectiveTypeHigherIsBetter,
MissionObjectiveTypeLowerIsBetter
};
```

When are missions generated?

When no missions are left in the `missions` array, filled by the developer, the method `+ (Mission *)generateNewMission` is called, that generates a new mission, based on the ones created by the developer.

How can I create custom Missions?

Therefore you need to subclass the `Missions` class. Take a look at the 3 initially provided mission types, to get all implementation details.

Basically most missions will use these two methods:

```
- (void)missionStart:(Game *)game
{
    // capture the initial state, e.g. initial distance
    startDistance = game.meters;
}

- (void)generalGameUpdate:(Game *)game
{
    // check if goal is reached, e.g. the required distance
    if ((game.meters - startDistance) > runningDistance.goalValue)
    {
        self.successfullyCompleted = TRUE;
    }
}
```

For the mission generator it is important, that all your objectives are initialized in the `-(id)init` method:

```
- (id)init
{
    self = [super init];

    if (self)
    {
        self.missionObjectives = [[NSMutableArray alloc] init];

        // available mission objectives need to be defined here, for mission
generator
        killAmountOfEnemies = [[MissionObjective alloc]
initWithObjectiveType:MissionObjectiveTypeHigherIsBetter];
        [self.missionObjectives addObject:killAmountOfEnemies];

        withinDistance = [[MissionObjective alloc]
initWithObjectiveType:MissionObjectiveTypeLowerIsBetter];
        [self.missionObjectives addObject:withinDistance];
        // the goal value will be set later on, either manually or by the
mission generator
    }

    return self;
}
```

Basically there will be 4 steps to your first custom mission:

1. Create a subclass of `Mission`
2. Initialize your MissionObjectives in `-(id)init`
3. Implement `-(void)missionStart:` to capture the game variables, when your

mission starts.

4. Implement `-generalGameUpdate` to check, if your mission goals have been reached. Once that happens set `self.successfullyCompleted = TRUE;`

Why do I need to implement -(id)initWithCoder ?

The Missions are stored to disk when the game terminates. Therefore that method needs to be implemented. Your implementation probably will look similar to this one:

```
- (id)initWithCoder:(NSCoder *)decoder
{
    // call the superclass, that initializes a mission from disk
    self = [super initWithCoder:decoder];
    // additionally store a reference to the 'runningDistance' objective in
the instance variable.
    runningDistance = [self.missionObjectives objectAtIndex:0];

    return self;
}
```

You call `[super initWithCoder:decoder]`, because the `Mission` class will take care of encoding/decoding all the default fields. Only if you add anything else than the default fields, you will have to implement `initWithCoder`. All the initial Missions in the template game have an instance variable, to be able the reference the objectives more quickly. This relation needs to be setup in `initWithCoder:`.

How can the Mission System be extended?

Probably the next useful step would be to add further messages, that missions can receive. This would allow a wider variety of Missions. One example:

```
- (void)monsterKilled:(Class)monsterClass;
```

This method could provide the type of monster that has been killed and could be used by a totally new kind of mission.

Store

Currently the Store only supports the In-App-Purchase of Coins. These coins can be used for an 'Go On' and 'Skip Ahead' action.

How can I add StoreItems?

The Store is setup in `[Store setupDefault]`.

```
        CoinPurchaseStoreItem *item1 = [[CoinPurchaseStoreItem alloc] init];
        item1.title = @"40 Coins";
        item1.thumbnailFilename = @"missions_1.png";
        item1.detailDescription = @"Pocket Money!";
        item1.cost = 0.99f;
        item1.coinValue = 40;
        item1.productID = @"TestProductID";
        item1.inGameStoreImageFile = @"buy1.png";
```

…

Finally all items are added to the stores. The Game has two different Stores. The one is the InGameStore, which is presented, when the player attempts to buy an 'Skip Ahead' or 'Go On' action, but does not have enough Coins.

All items that shall be presented in the inGameStore need an `inGameStoreImageFile`.

To the normal store screen, the items are added in different sections. The sample code does that in the `setup` dictionary.

```
        inGameStoreItems = [@[item1, item2, item3] mutableCopy];
        NSDictionary *setup = @{@"Coins": @[item1, item2,item3, item4,
item5, item6]};
```

To add new Coin-Items, simply create new instances and add them to both or one of the two stores.